The Report Committee for Iat Pui Chan
Certifies that this is the approved version of the following report:

USB 2.0 ULPI Design

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Jacob Abraham

Mark McDermott

USB2.0 ULPI Design

by

Iat Pui Chan, BSEE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2009

Abstract

USB2.0 ULPI Design

Iat Pui Chan, MSE

The University of Texas at Austin, Fall 2009

Supervisor: Jacob Abraham

This report outlines an implementation of an USB 2.0 ULPI LINK design. Chapter 1 presents an overview of the design and hardware required in the project. Chapter 2 presents how the design functions in an USB system. Chapter 3 describes the hardware implementation. Simulation result and synthesis result are shown in Chapter 4 and 5.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The USB2.0 design implementation covered in this report involves both hardware and software portion. The hardware portion is the USB 2.0 design itself synthesized and tested on the LL5000Baseboard. The software portion involves a simple Linux-based USB driver to handle RX interrupt to allow data bandwidth measurement.

USB stands for universal serial bus. USB2.0 supports three speed modes, a high speed mode of 480Mbits/s, a full speed mode of 12Mbits/s, and a low speed mode of 1.5Mbits/s. USB 1.0 only supports full speed and low speed mode. USB2.0 is backward compatible with USB1.0.

1.1 SYSTEM OVERVIEW

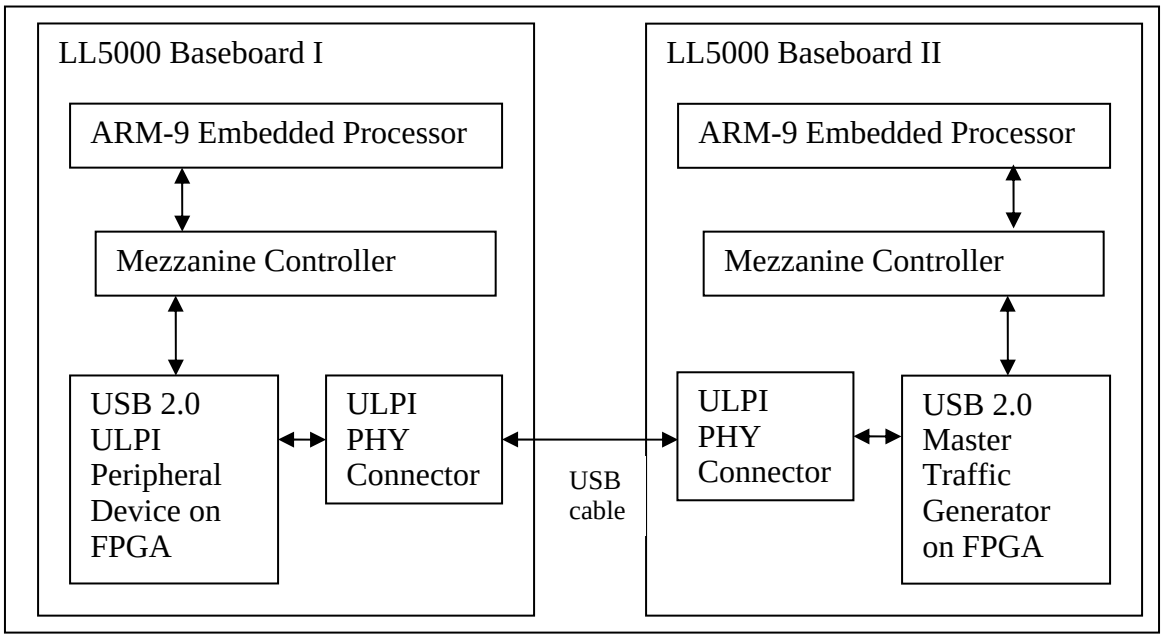


Figure 1.1: System Overview

System implementation can be divided into hardware and software portion. Software portion involves a simple Linux-based USB driver to handle RX interrupt to allow data bandwidth measurement. The hardware design consists of two identical LL5500 Baseboards as shown in Figure 1.1. One of the boards will act as USB 2.0 ULPI peripheral device and the other as USB 2.0 master traffic generator.

UTMI stands for USB 2.0 Transceiver Macrocell Interface. It is the interface supported by UTMI core design. ULPI stands for UTMI Low Pin count Interface. It allows a lower cost design as using lower pin count ICs. Typical UTMI design requires a package of 64 to 80pins whereas ULPI only 32 pins or less.

1.1 LL5000 DEVELOPMENT BOARD

For this project concern, the LL5000 development board has the following main components: ARM-9 embedded processor and mezzanine controller inside the TTL6219 development system, the Xilinx Spartan 3000 and the ULPI PHY connector from the Baseboard.

1.2 SOFTWARE DESIGN

Software design developed for this report is mainly USB2.0 Linux-based driver; code is executed by the ARM core. Software driver written for master traffic generator will initiate the hardware to transmit the data packet and transmit across to the USB slave. The driver implemented for the peripheral device is to timestamp data receive interrupt. This allows future calculation of data bandwidth.

1.3 USB 2.0 ULPI PERIPHERAL LINK

The ULPI peripheral Link design acts as a USB slave in the system, and is synthesized on the TLL5000 development board. The design asserts receive interrupt

when data packet is received from the host. Software driver is written to handle the interrupt to calculate the bandwidth.

1.4 USB2.0 MASTER TRAFFIC GENERATOR

USB2.0 Master Traffic Generator acts as a USB host in the system. It is an identical design of the peripheral module synthesized on another TLL5000 Baseboard. The traffic generator generates data traffic in order to verify data bandwidth of USB 2.0 ULPI design. This is done by utilizing a combination of software and the capability of ULPI test generator logic to emulate an USB2.0 host device.

1.5 ULPI PHY CONNECTOR

The ULPI PHY is provided by the TLL5000. Each PHY device is bonded out to a TYPE A USB connector. Figure 1.2 shows the physical connector interface with 4 shielded wires. Two of them are for twisted pair differential data signals, and two for power (+5v & Gnd).

Pins Number	Function
1	Vbus (5 Volts)
2	D-
3	D+
4	Gnd

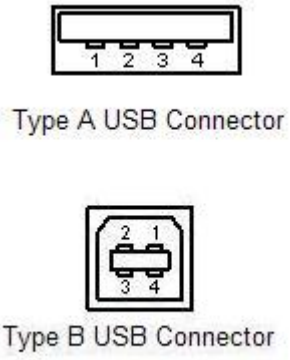


Figure 1.2 Pin Layout of USB connector [1]

Chapter 2: ULPI Peripheral Link Operation

This section illustrates how ULPI Peripheral Link and its operation are applied in the system. Figure 2.1 is a system view with USB Host and USB slave devices in the system. Our ULPI Peripheral Link design could be one of these USB slave devices. USB supports four types of transfer operation: Control Transfer, Interrupt Transfer, Isochronous Transfer, and Bulk Transfer. Each transfer requires implementation of software and hardware. Software design covered in this report can only support control and bulk transfer functions. However, the hardware is implemented to support all USB functions.

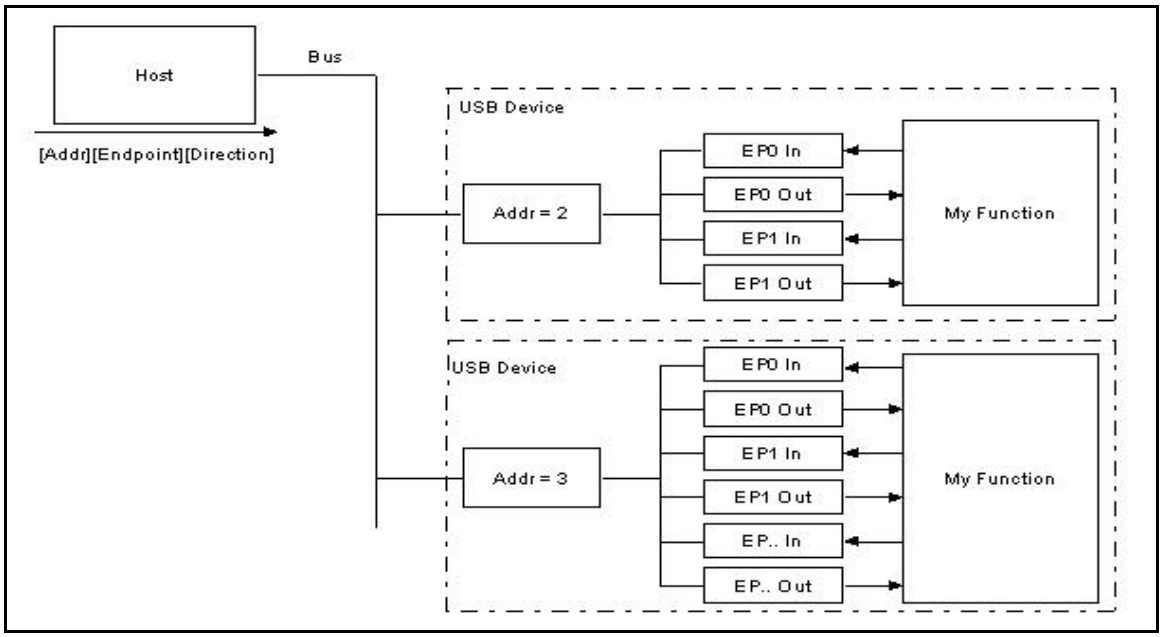


Figure 2.1 ULPI Peripheral Link System Overview [1]

2.1 USB BUS TRANSFER OPERATION

This subsection illustrates how an USB host can address our ULPI Link design. Figure 2.2 shows the USB Control Transfer operation in the system. Figure 2.3 shows the content of a USB packet in different operation stages. Each packet has a PID field to indicate the packet types. During the first phase of each USB operation stage, the USB packet transmitted by the USB host has an address field and an endpoint field.

The host communicates with the ULPI Peripheral device through address matching in its function core logic. This address value will be compared to the value of USB slaves' functional address register. If a match occurs, the slave will be addressed. The endpoint field indicates which endpoint register will be addressed.

The design is implemented with 4 endpoints and can easily be extended to support up to 16 endpoints. Each endpoint is implemented with a set of endpoint registers to serve as either EP_OUT or EP_IN. Refer to Figure 2.1. These registers contain address pointers to where to fetch outgoing data or to store incoming data. In the DATA Stage of Figure 2.2, these registers control Buffer 1 where to source data from and when to generate buffer empty interrupt. In the Status Stage of Figure 2.2, they control Buffer 0 where to sink its data to and when to generate buffer full interrupt.

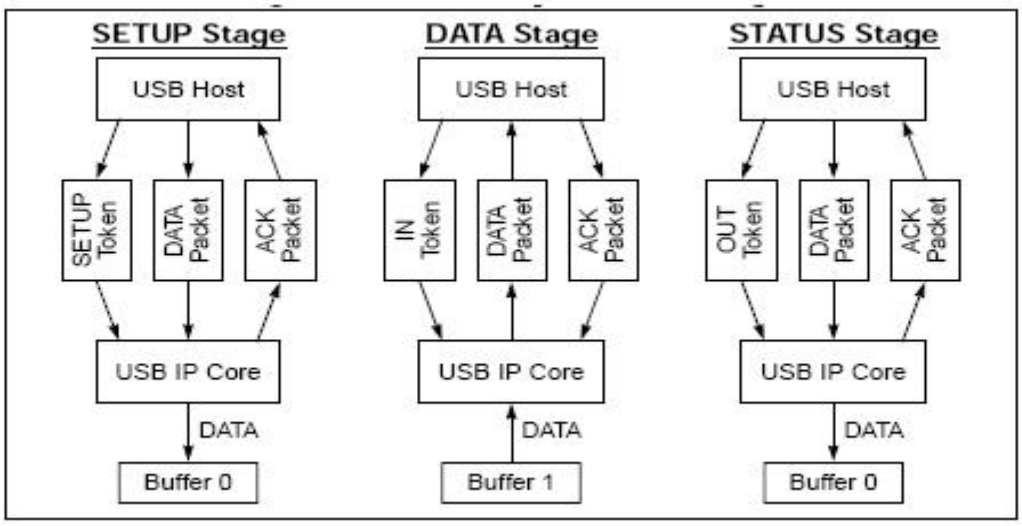


Figure 2.2 USB Operation Stage [2]

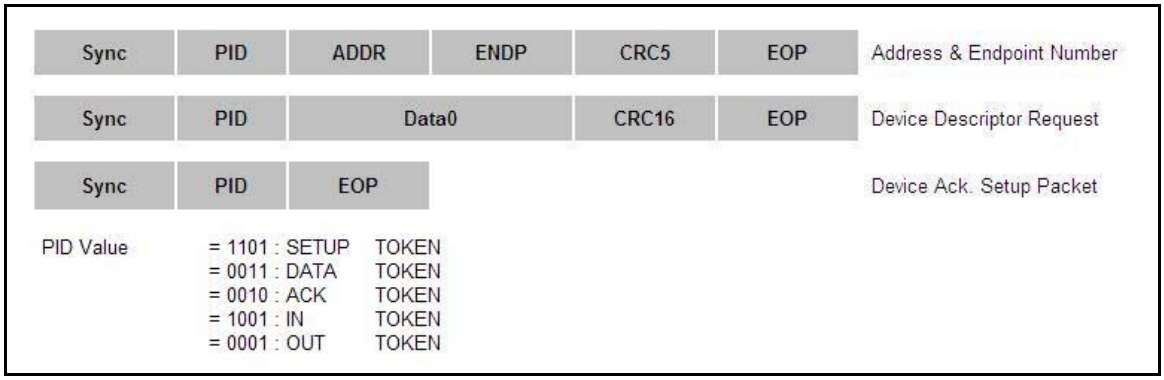


Figure 2.3 USB Packet Content [1]

2.2 UTMI & ULPI INTERFACE

Figure 2.3 illustrates where the wrapper logic fits into the system. The ULPI Link includes UTMI+ Core and ULPI Link wrapper logic. The ULPI Link is synthesized in FPGA of LL5000 Baseboard. ULPI PHY is a device already available on the Baseboard.

Each USB transfer will go through a ULPI PHY device. The ULPI PHY device will translate incoming USB bus data to ULPI receive commands and ULPI transmit commands to USB data. Between the ULPI PHY connector and the UTMI+ Core logic,

an ULPI Link wrapper is required to translate the ULPI command to the UTMI core logic.

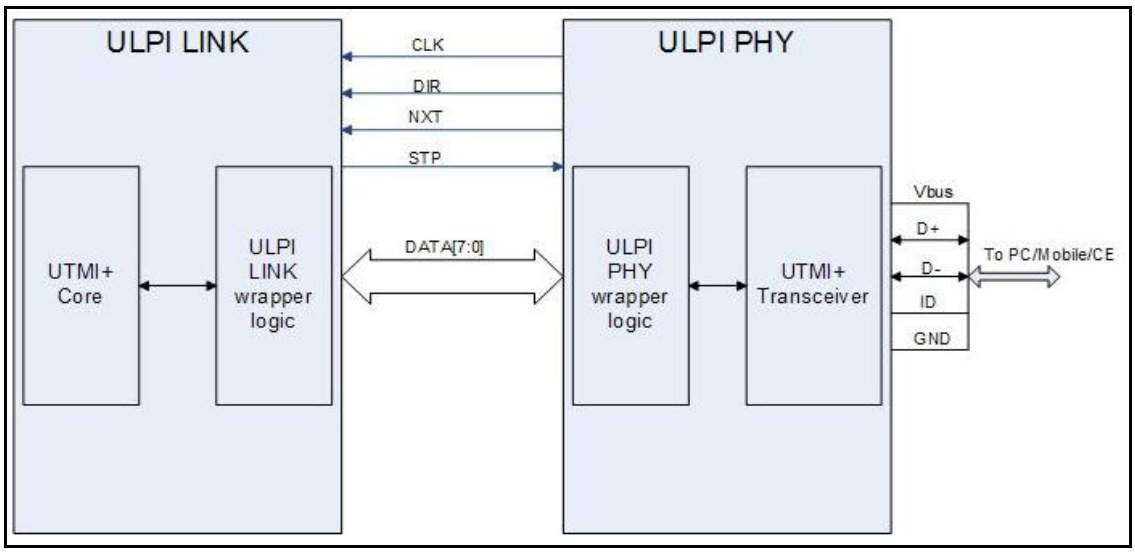


Figure 2.4 ULPI Link Design Overview [3]

2.3 ULPI COMMAND

There are two types of ULPI commands supported in the design. One type of ULPI commands accesses a register on the ULPI PHY device. As shown in Figure 2.5, the ULPI link design initiates register read/write command to the ULPI PHY register address mapped in the system.

The other type is UTMI-to-ULPI command. A transmit command shown in Figure 2.6 is issued from ULPI to ULPI PHY connector when ULPI peripheral function core transmits data to the host. A receive command shown in Figure 2.7 is issued from ULPI PHY connector to ULPI when ULPI PHY receives USB data from the host.

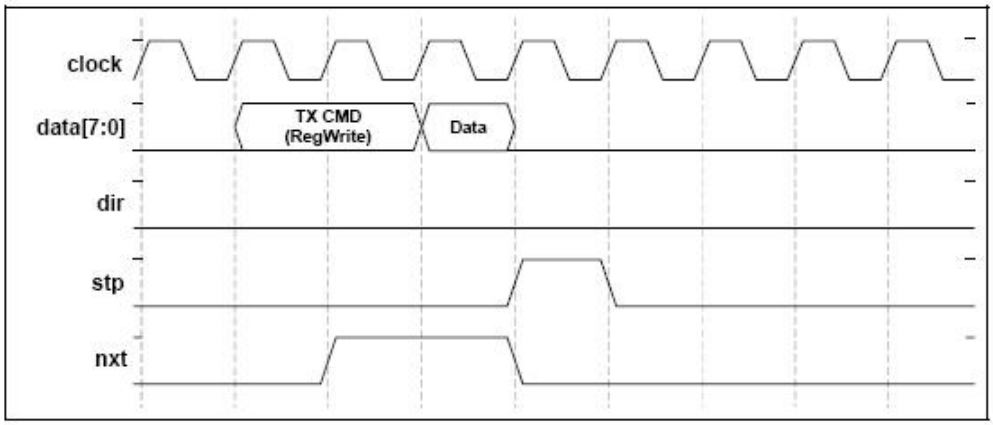


Figure 2.5 ULPI Register Write Command [3]

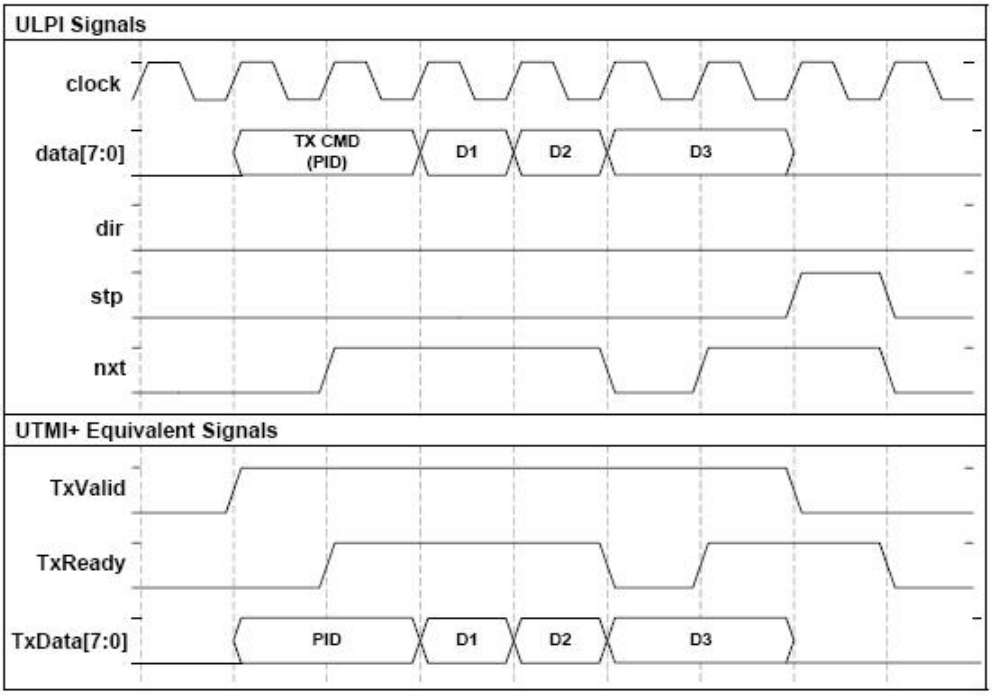


Figure 2.6 UTMI-to-ULPI Transmit Command Translation [3]

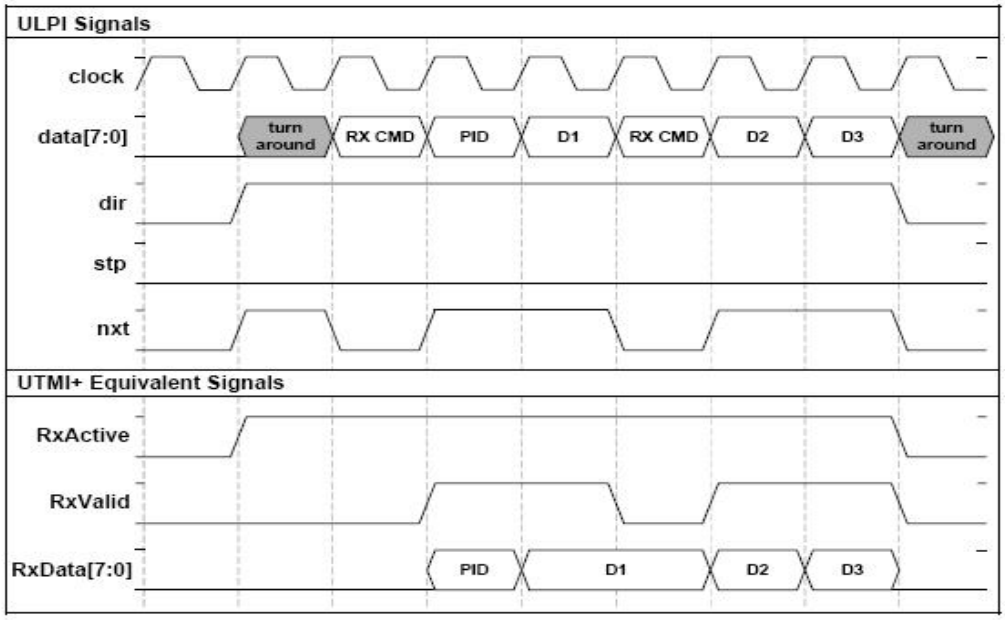


Figure 2.7 UTMI-to-ULPI Receive Command Translation [3]

2.4 ULPI PHY TO ULPI CORE OPERATION

This section contains a step-by-step example to illustrate how ULPI operation fits into the system. The following steps show how ULPI receives and transmits commands applied during USB Data stage as shown in Figure 2.2.

IN-Token Step

- USB host initiates the transaction by sending an IN token package.
- The package will transmit through the ULPI PHY. ULPI PHY will translate the packet into ULPI Receive command.
- ULPI decoder will decode and translate the Receive command into UTMI signals.
- USB peripheral core decodes the UTMI signals in its protocol layer and compares against its endpoint register for an address match.

Data Packet

- Upon receiving USB In-Token, the USB peripheral device responds by transmitting data packet out.
- UTMI core asserts the corresponding UTMI signals to transmit the data packet.
- The data packet is translated from UTMI signals into ULPI Transmit command.
- ULPI PHY connector receives the command and drives the USB bus accordingly.

Acknowledge

- Upon USB host receiving data packet, USB host responds by sending an Acknowledge signal.
- ULPI PHY translates the packet into ULPI Receive command.
- ULPI decoder translates the Receive command into UTMI signals.
- UTMI core receives the Acknowledge packet and completes the DATA stage cycle.

Chapter 3: Hardware Implementation

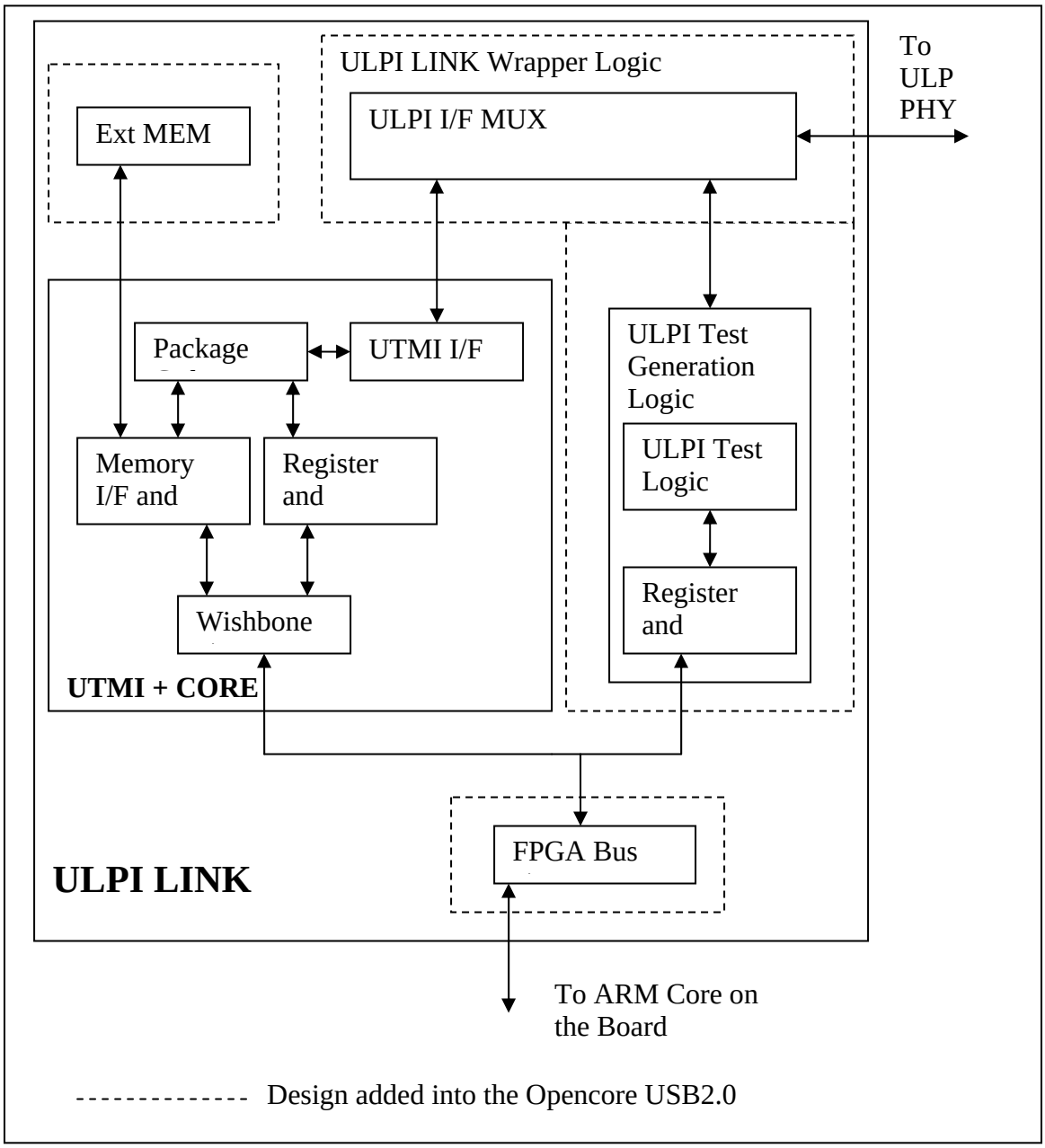


Figure 3.1 USB 2.0 Peripheral Design Block Diagram

This section describes each component of the USB Link design in the platform. USB2.0 ULPI peripheral device is based on an existing USB 2.0 slave design from OpenCore.org. An UTMI-to-ULPI wrapper is implemented in order to allow the design to communicate with the ULPI connector on the board. Figure 3.1 illustrates the design from the USB Opencore.org and additional blocks implemented in this project.

3.1 USB 2.0 UTMI + CORE

The design is an open source IP downloaded from Opencore.org. Figure 3.2 is the diagram directly extracted from Opencore USB2.0 core specification. Part of the design is modified in order to meet timing requirements of the external memory (memory from the FPGA). The PHY in Figure 3.2 is the ULPI PHY connector on the LL5000 Baseboard. The wishbone interface is connected to the function microcontroller through the Mezzanine bus IF.

Clock domain 1 is clocked at 60Mhz. The clock is driven by PHY CLK from ULPI PHY connector. Clock domain 2 is clocked at 64 MHz by CPLD logic block. That is the same clock driving the Mezzanine bus I/F logic.

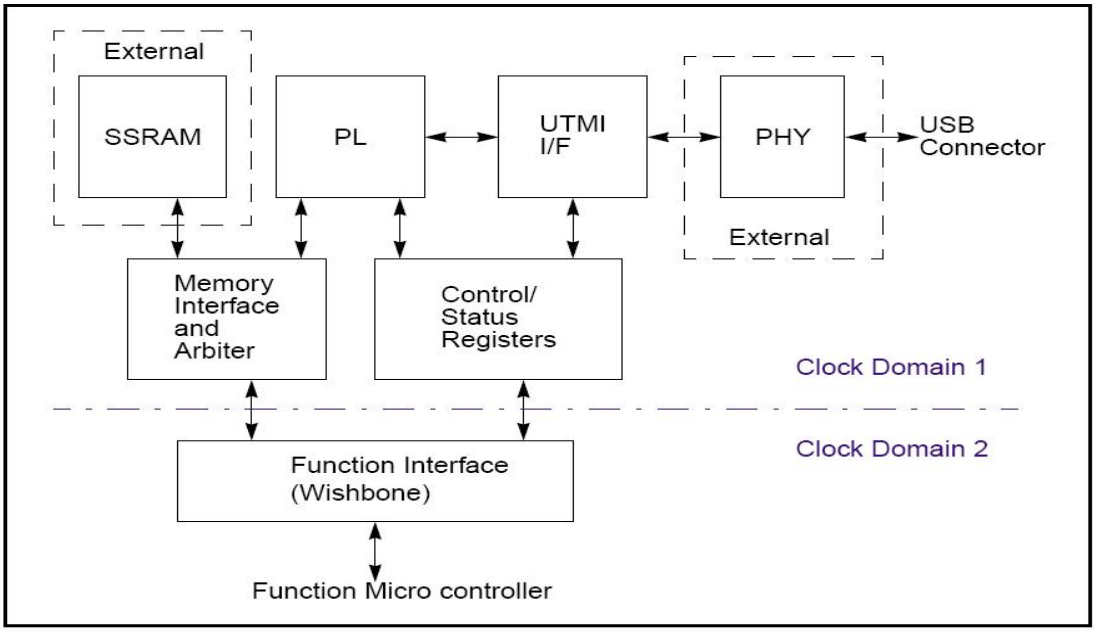


Figure 3.2 UTMI+Core Design[2]

3.2 ULPI LINK WRAPPER LOGIC

The design can be divided into UTMI-to-ULPI Translator logic, ULPI Test Generation logic, and ULPI I/F Mux. Figure 3.3 shows the detailed design of the ULPI Link Wrapper Logic.

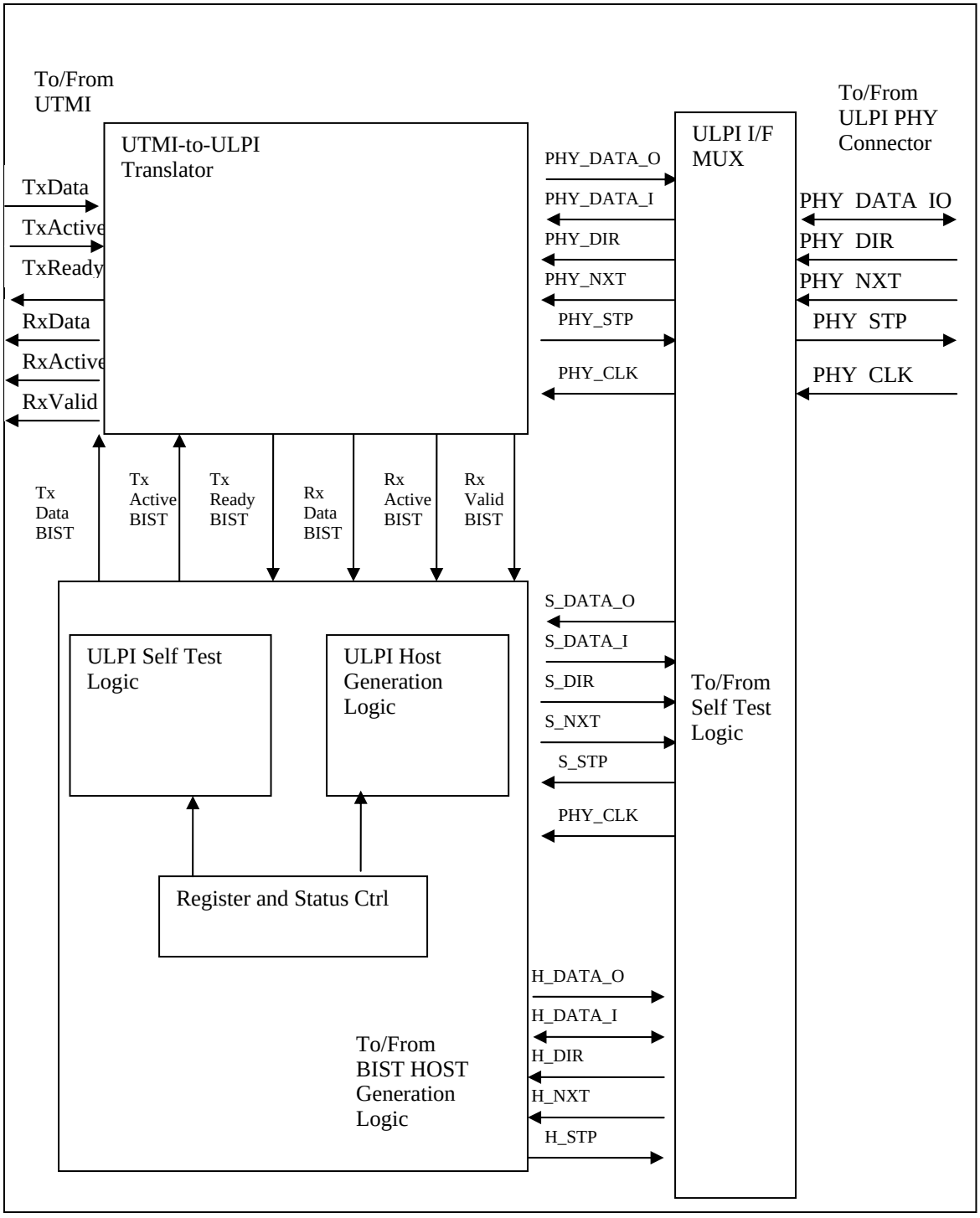


Figure 3.3 ULPI Link Wrapper

3.2.1 UTMI-to-ULPI Translator Logic

The design has the following two functions:

1. It translates UTMI TX and RX commands to corresponding ULPI commands.
2. It translates UTMI TX and RX commands from ULPI test generation logic to ULPI PHY Register RW command.

3.2.2 ULPI TEST GENERATION LOGIC

The design acts as a built-in self test to verify major functions of the ULPI Link design on the board. It also has a 32-bytes deep stack to track the first 32-bytes data received from the ULPI PHY connector.

Figure 3.4 shows the ULPI commands involved in each test logic function.

1 The design handles any mezzanine bus R/W to any ULPI PHY connector registers. It translates mezzanine bus access and generates ULPI commands in order to read/write to the registers in the ULPI PHY connector.

2 The design acts as an ULPI Link device. It generates ULPI TX commands to allow test generator functioning as a USB host. Thus, ULPI Link can receive data packets from the USB host and allows measuring data bandwidth of the system,

3 The design acts as an ULPI PHY Connector. It generates ULPI RX commands to USB2.0 ULPI in order to verify USB2.0 ULPI device that can decode RX command properly.

4 The design acts as an ULPI PHY Connector. It responds to ULPI TX commands from USB2.0 ULPI and verifies that the USB2.0 ULPI transmits ULPI command out properly.

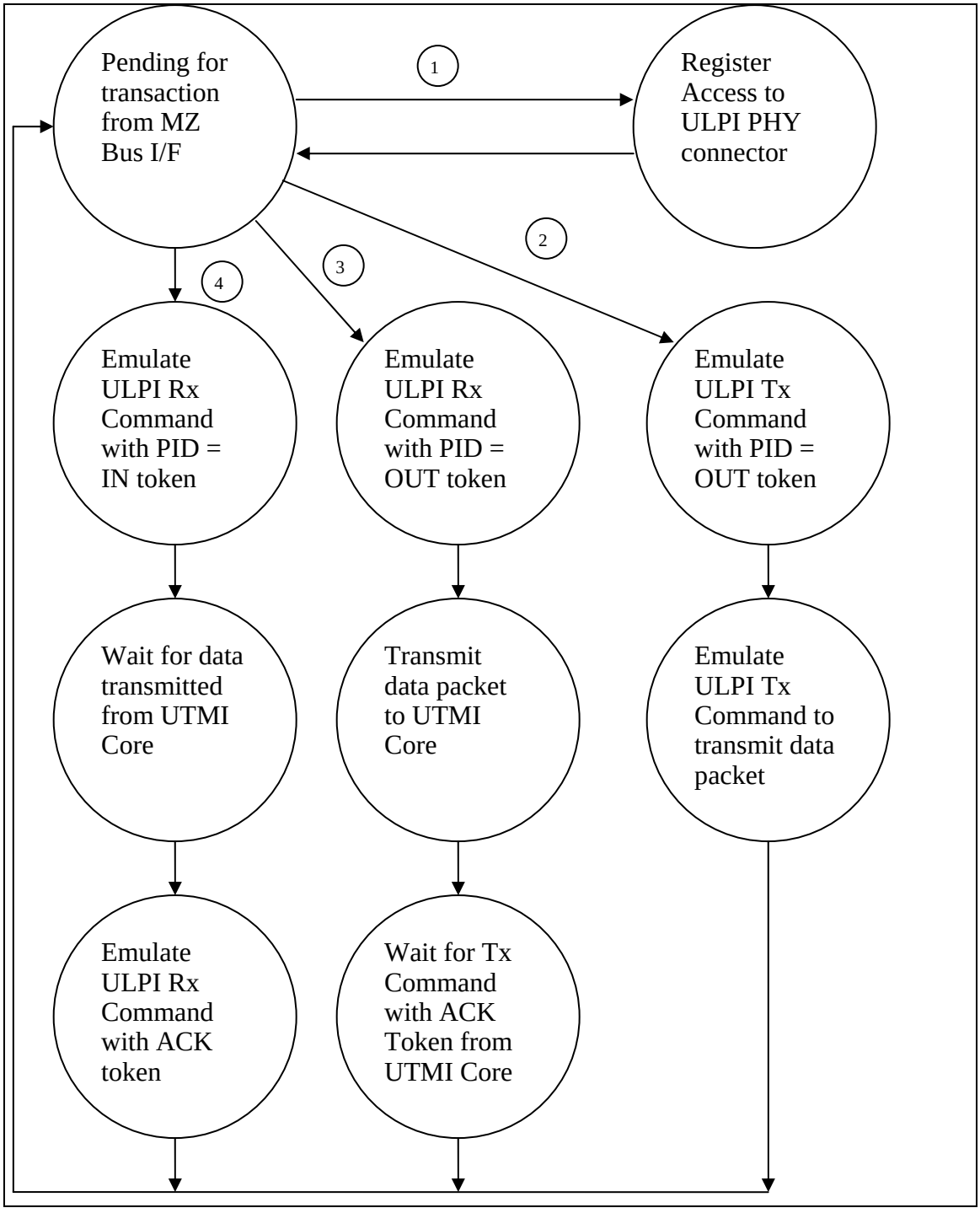


Figure 3.4 ULPI Command Utilized in ULPI Test Logic

3.2.3 ULPI I/F MUX

The design has the following three functions:

1. When the ULPI Test generator is not acting as LINK or PHY test device, it routes the signals from the ULPI PHY connector to the UTMI-to-ULPI Translator logic block.
2. When the ULPI Test generator acts as the PHY device, it routes the signals from the ULPI Test generator to the UTMI-to-ULPI Translator logic block.
3. When the ULPI Test generator acts as the LINK device, it routes the signals from the ULPI Test generator to the ULPI PHY Connector.

The initial design failed in the timing of signals PHY_DATA_IO and PHY_STP to their corresponding pins. To allow better timing, the design has gated signals PHY_DATA and PHY_STP before sending out to the pins of the FPGA. The ULPI I/F Mux prefetches transmit data (PHY_DATA) and control signal (PHY_STP) from the UTMI device to allow correct functionality.

3.3 MEZZANINE BUS I/F

The design decodes bus read and write access from the microcontroller on the board. Refer to Figure 3.5 LL5000 Baseboard Software Memory Map and Table 3.1. Bus access from the microcontroller will route to the ULPI Link, the ULPI test logic, and the external memory based on the address range.

3.4 EXTERNAL MEMORY

The design is a memory model available in the Xilinx FPGA. As shown in Figure 3.2, the external memory is accessible through both ULPI interface and wishbone bus interface. Based on hardware verification, the memory model requires 2 cycles to read or write to operate properly.

3.5 MEMORY MAP

Figure 3.5 is the software memory map of the LL5000 Baseboard system. Table 3.0 is the memory map implemented in the ULPI Link design.

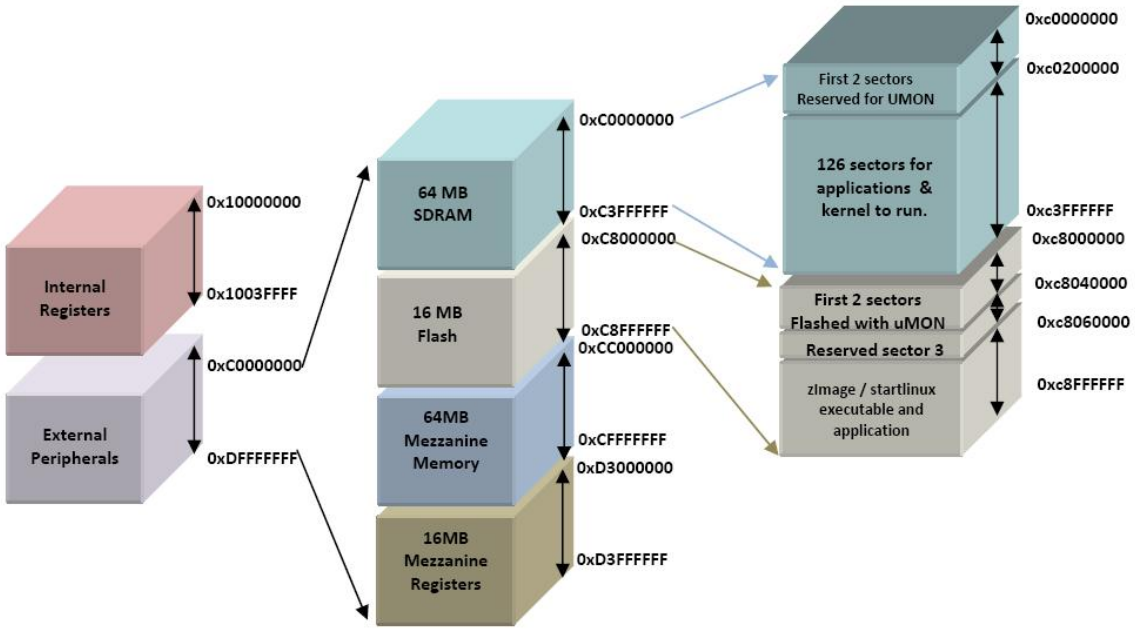


Figure 3.5 LL5000 Baseboard Software Memory Map [4]

Table 3.1 ULPI Link Memory Map

Memory Space Description	System Address Range
External Memory	0xD3000000-0xD303FFFF
UTMI Core	0xCC000000-0xCC000800
ULPI PHY Connector	0xCC000800-0xCC000824
ULPI Test Generation Logic	0xCC000900-0xCC000908

Table 3.2, 3.3 and 3.4 provide a description of registers implemented in the ULPI design. Detail UTMI Core and ULPI PHY connector register description can be found from the following two sites:

www.opencores.org[2]

www.ulpi.org/documents.html[3]

Registers in UTMI Core and ULPI Test generation logic are implemented with 32-bit register. ULPI PHY connector registers are 8 bit. To simplify the design, a 32-bit register mapping is applied on ULPI PHY registers. In other words, a mezzanine 32-bit register write to ULPI PHY register (e.g. 0xCC00808) will still be interpreted as 8-bit write to the ULPI PHY connector register (0x07).

Table 3.2 UTMI Core Registers

Base address = 0xCC00000

Address	Register Name	Description
0x00	CSR	Control Status Register
0x04	FA	Function Address
0x08	INT_MSK	Interrupt Mask for Endpoint Independent Source

Address	Register Name	Description
0x0C	INT_SRC	Interrupt Source Register
0x10	FRM_NAT	Frame Number and Time
0x14	UTMI_VEND	Vendor Specific I/O Port
0x40	EP0_CSR	Endpoint 0: CSR
0x44	EP0_INT	Endpoint0: Interrupt
0x48	EP0_BUF0	Endpoint0: Buffer Register 0
0x4C	EP0_BUF1	Endpoint0: Buffer Register 1
0x50	EP1_CSR	Endpoint 1: CSR
0x54	EP1_INT	Endpoint 1: Interrupt
0x58	EP1_BUF0	Endpoint 1: Buffer Register 0
0x5C	EP1_BUF1	Endpoint 1: Buffer Register 1
0x60	EP2_CSR	Endpoint 2: CSR
0x64	EP2_INT	Endpoint 2: Interrupt
Address	Register Name	Description
0x68	EP2_BUF0	Endpoint 2: Buffer Register 0
0x6C	EP2_BUF1	Endpoint 2: Buffer Register 1
0x70	EP3_CSR	Endpoint 3: CSR
0x74	EP3_INT	Endpoint 3: Interrupt
0x78	EP3_BUF0	Endpoint 3: Buffer Register 0
0x7C	EP3_BUF1	Endpoint 3: Buffer Register 1

Table 3.3 ULPI PHY Connector Registers

Base address = 0xCC00800

Address	Register Name	Description
0x00	VID	Vendor ID. Read/Write to ULPI register address 0x00
0x04	FCR	Function Control Register. Read/Write to ULPI register address 0x04
0x08	ICR	Interrupt Control Register. Read/Write to ULPI register address 0x07
0x0C	OTGCR	OTG Control Register. Read/Write to ULPI register address 0x0a
0x10	IERR	USB Interrupt Enable Rising Register. Read/Write to ULPI register address 0x0d
0x14	IEFR	USB Interrupt Enable Falling Register. Read/Write to ULPI register address 0x10
0x18	ISR	Interrupt Status Register. Read/Write to ULPI register address 0x13
0x1C	ILR	Interrupt Latch Register. Read/Write to ULPI register address 0x14
0x20	DGR	Debug Register. Read/Write to ULPI register address 0x15
0x24	SR	Scratch Register. Read/Write to ULPI register address 0x16

Table 3.4 ULPI Test Generation Logic Registers

Base address = 0xCC00900

Address	Register Name	Description
0x00	ULPI_TG_CTRL	Register controls the action of ULPI Test generation logic. Bit Field[31] = Status bit when ULPI_TG acts as PHY Bit Field[20] = Status bit when ULPI_TG act as LINK Bit Field [27:24] = EndP sent for self test packet Bit Field [22:16] = Addr for self test packet Bit Field [11:8] = Token ID for self test packet Bit Field [2] = Host Pattern Generation Bit Field [1] = SELF Test Enable Bit field [0] = ENABLE
0x04	ULPI_TG_STATUS	Register contains the first four bytes of data received from an IN-Token self test.
0x08	ULPI_TG_SEED	Register control the data packet being transmitted in OUT-Token self test or Host data pattern generation. Bit field [31:16] is the number of bytes to be transmitted. Bit field [7:0] is being utilized as the seed for data packet generation.
0x10	ULPI_TG_RX_CMD	Register pops the latest 4 bytes data from the received data stack.

Chapter 4 Hardware Simulation

This section describes testbench and simulation results of the design. The testbench allows basic functional verification of the design prior synthesis. As shown in Figure 4.1, the ULPI Link design is being instantiated twice. One acts as an USB slave device and the other as ULPI master generation Logic.

Unlike the actual system, the ULPI PHY connector is being replaced by a PHY model. The host ULPI PHY model and slave ULPI PHY model are connected through an ULPI Host-to-Slave PHY Model. The model performs a similar function as the actual ULPI PHY connectors. Thus, it can

- Respond to ULPI register R/W commands issued from ULPI LINK design
- Respond to UTMI-to-ULPI command and then generates correspond commands to the other side of the USB party. For example, when an ULPI Slave sends a ULPI TX command, the Slave PHY model firstly acknowledges the ULPI Slave. Then, the Slave PHY model forwards the command to Host PHY model. Finally, the Host PHY model generates RX command to the ULPI Host.

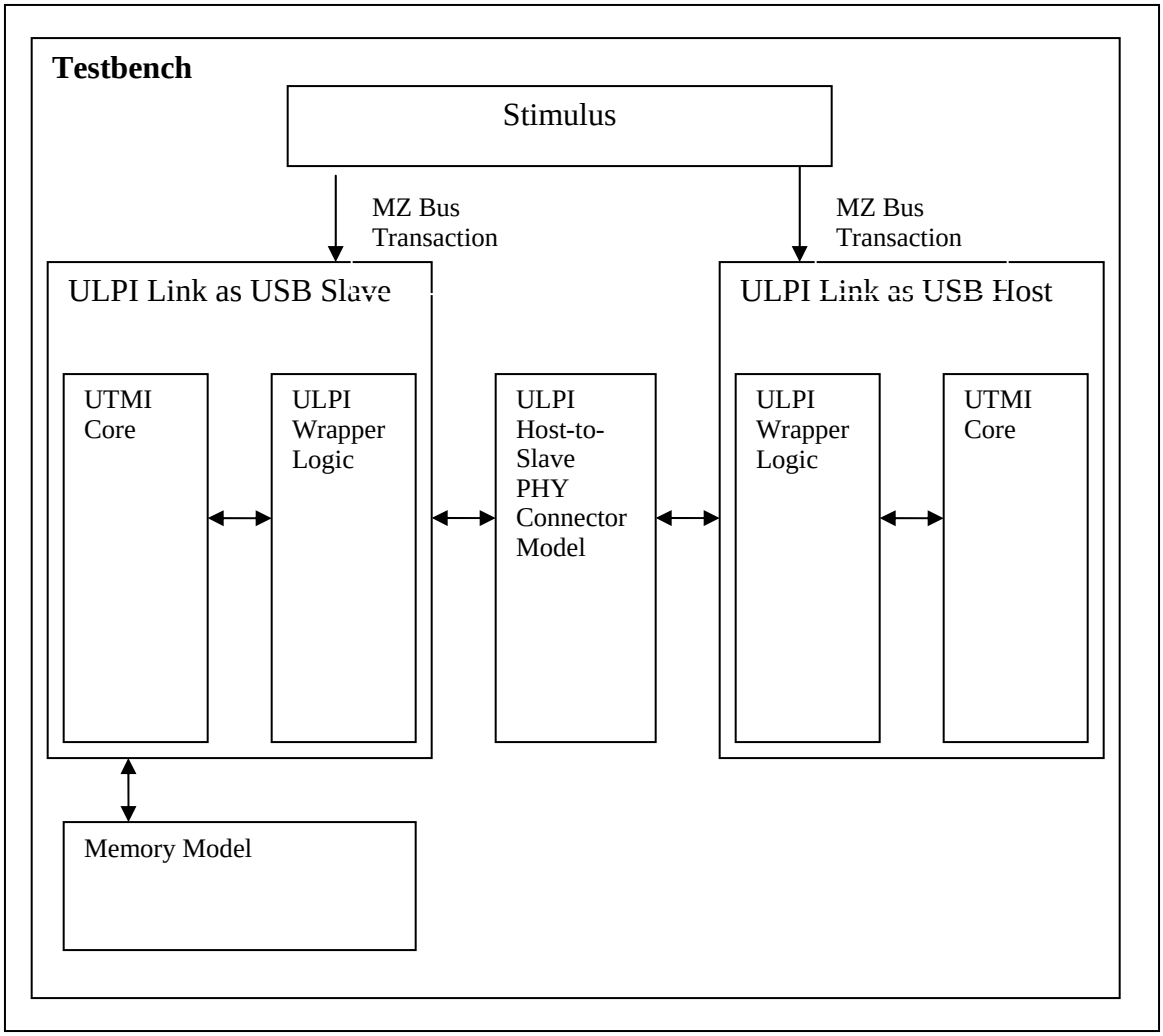


Figure 4.1 ULPI Link Testbench

4.1 SIMULATION RESULT OF FPGA I/F ACCESS

Figure 4.2 is the simulation results of register read/write to UTMI core function. The mezzanine bus controller asserts “MZ_CPLD_AS” and “MZ_CPLD_MISC7” to indicate each register read/write access to ULPI LINK design.

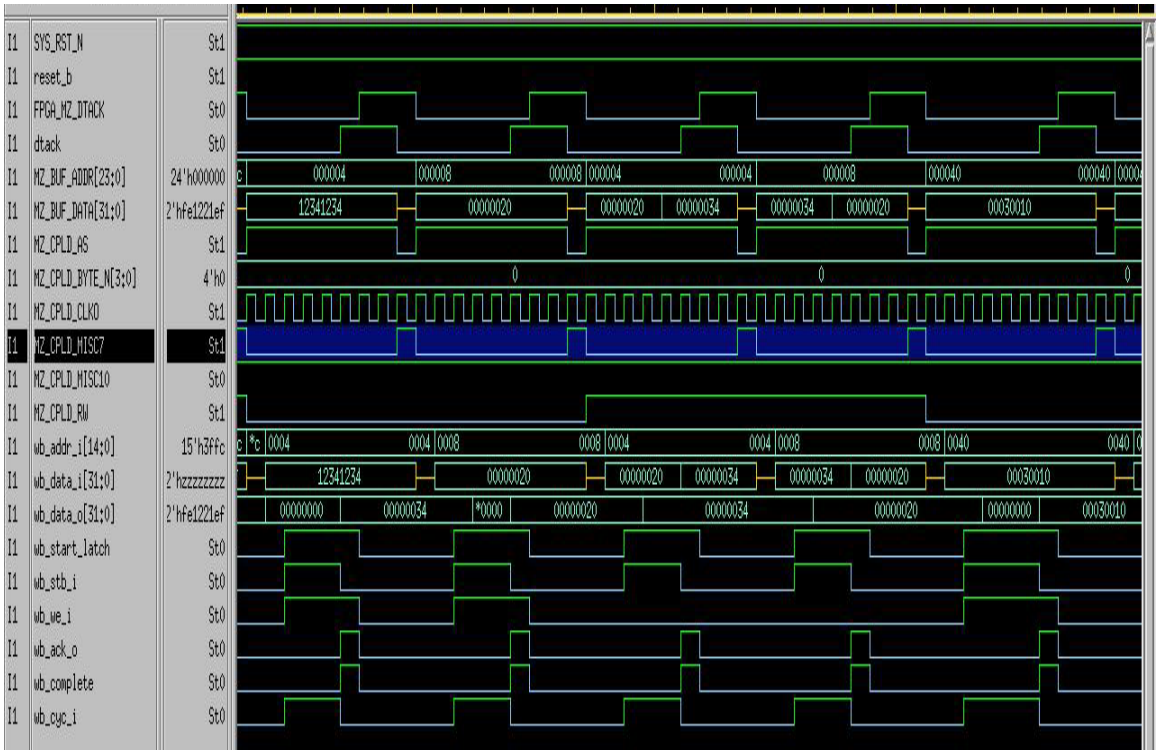


Figure 4.2 Mezzanine Bus Register Read/Write Simulation

4.2 SIMULATION RESULT OF MEMORY ACCESS

Figure 4.3 and 4.4 shows a memory write from MZ Bus I/F and UTMI I/F. The following steps will go through a typical write access.

1. The mezzanine bus controller asserts “MZ_CPLD_AS” and “MZ_CPLD_MISC10” to indicate a memory write access.
2. MZ bus signals get translated to wishbone bus signals. Thus, a wishbone write will assert to the ULPI LINK design.
3. ULPI LINK asserts a memory write to the external memory and asserts “wb_ack_o” signal to acknowledge to the wishbone write access.
4. MZ Bus negates MZ_CPLD_AS and bus access completes.

Figure 4.5 is PHY write to memory. Note the memory write is initiated when the ULPI LINK received a ULPI Rx command from the ULPI PHY connector.

Figure 4.3 Memory Write Simulation

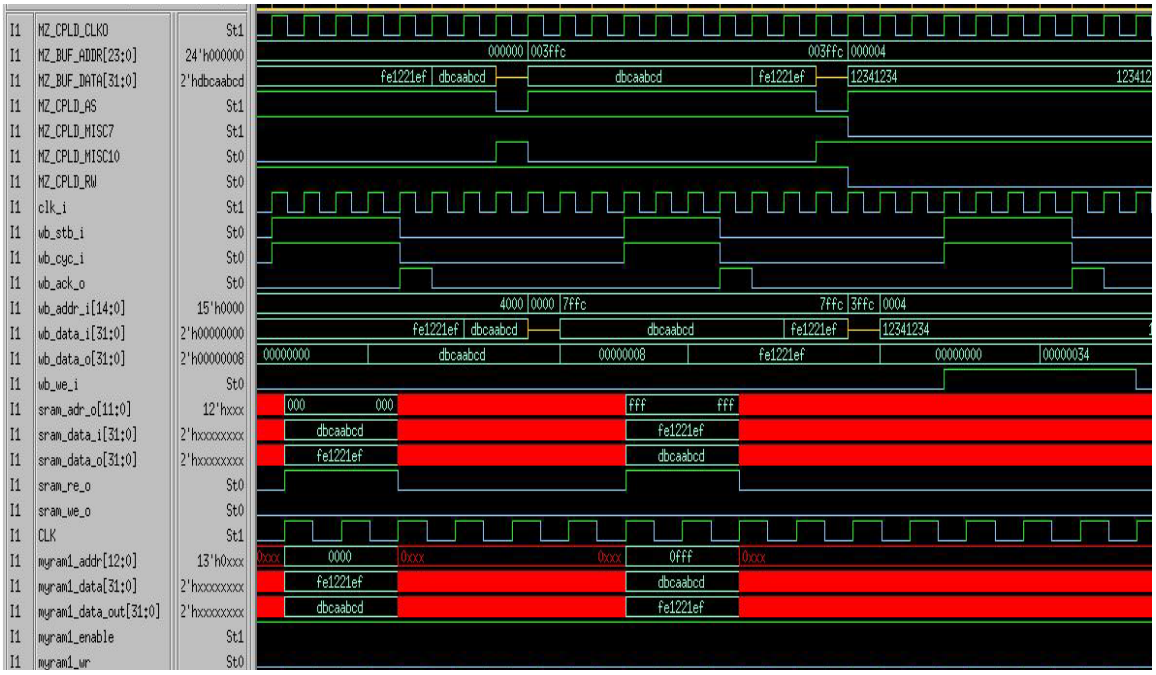


Figure 4.4 Memory Read Simulation

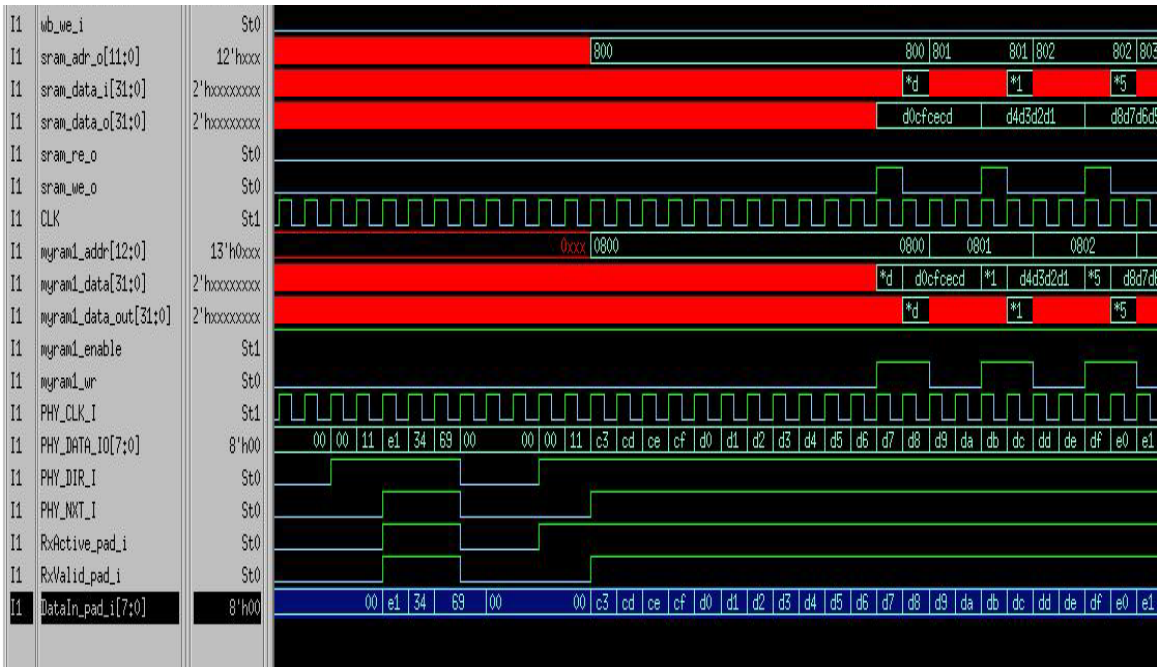


Figure 4.5 PHY Memory Write

4.3 SIMULATION RESULT OF ULPI COMMAND ACCESS

Figure 4.6 is ULPI register write and read command and illustrates how FPGA Bus I/F access triggers ULPI register I/F access.

1. ULPI register read/write access starts from MZ bus read/write
2. The design will translate the mezzanine bus read/write access to register read/write access to the USB test generation logic design.
3. The USB Test Generation logic will assert UTMI signals to trigger an USB register write command to the ULPI PHY connector.
4. The ULPI PHY connector acknowledges the register write by asserting the PHY_NXT.
5. The USB Test Generation asserts the “bist_reg_ack”.
6. “MZ_CPLD_AS” is negated and bus access completes.

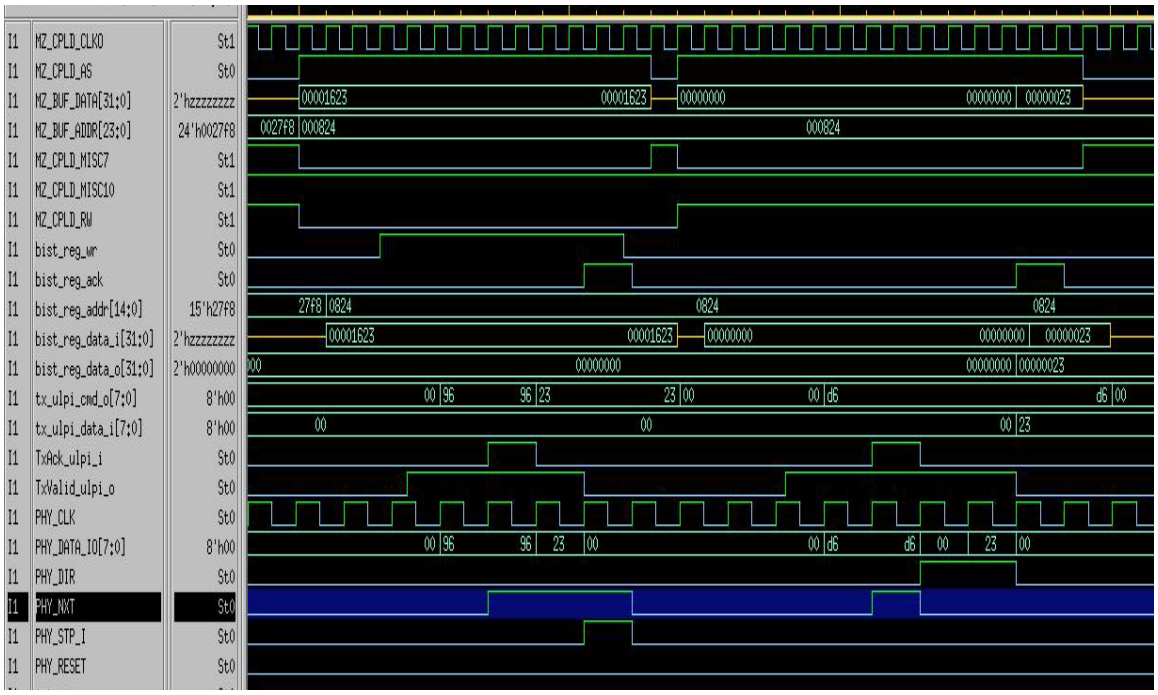


Figure 4.6 ULPI Register Write Access

4.4 SIMULATION RESULT OF SELF-TEST LOGIC: IN-TOKEN AND OUT-TOKEN TEST

Figure 4.7 and Figure 4.8 illustrate the operation of an USB Data Stage sequence. Refer to Figure 2.2 on Data Stage operation.

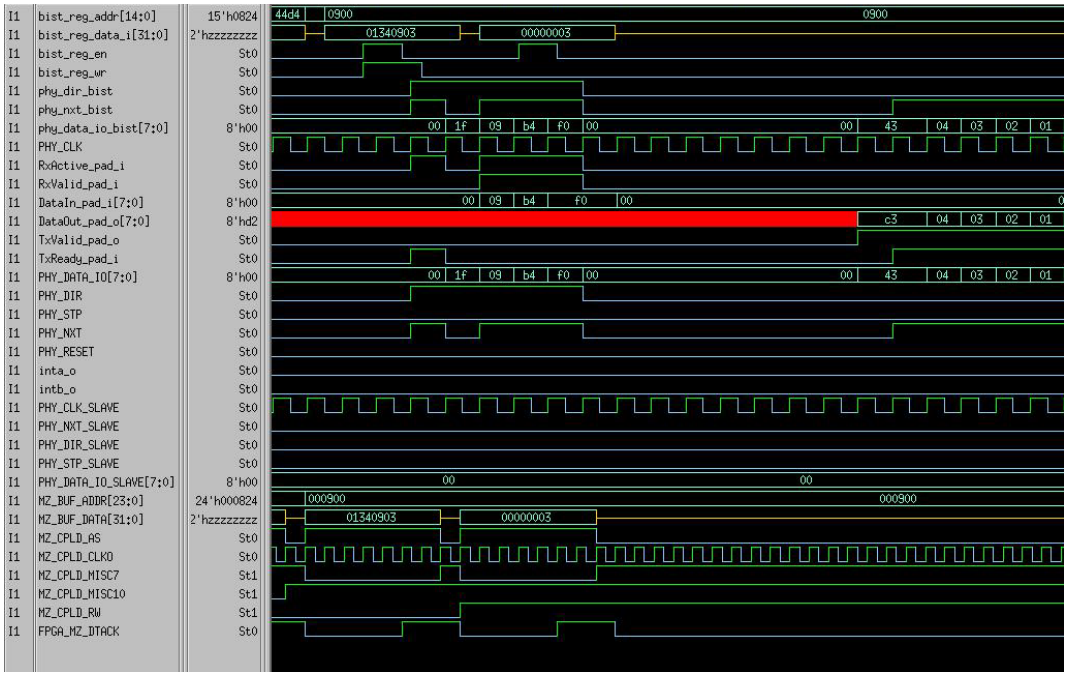


Figure 4.7 Data Stage Operation: IN Token & DATA Token

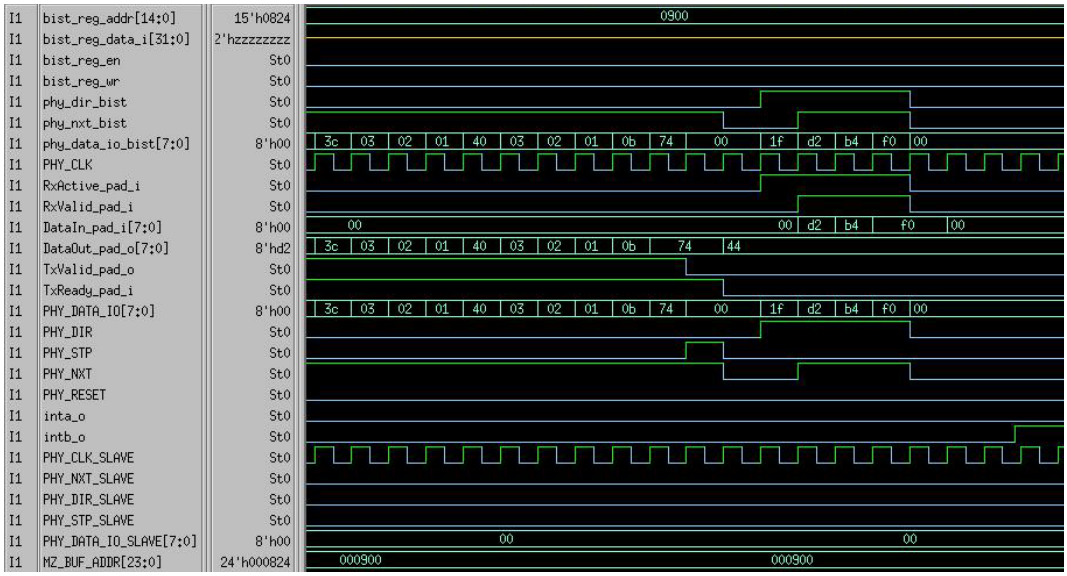


Figure 4.8 Data Stage Operation: DATA Token & ACK Token

Figure 4.9 and 4.10 shows the operation of an USB Status stage. Refer to Figure 2.2 on Status Stage operation.

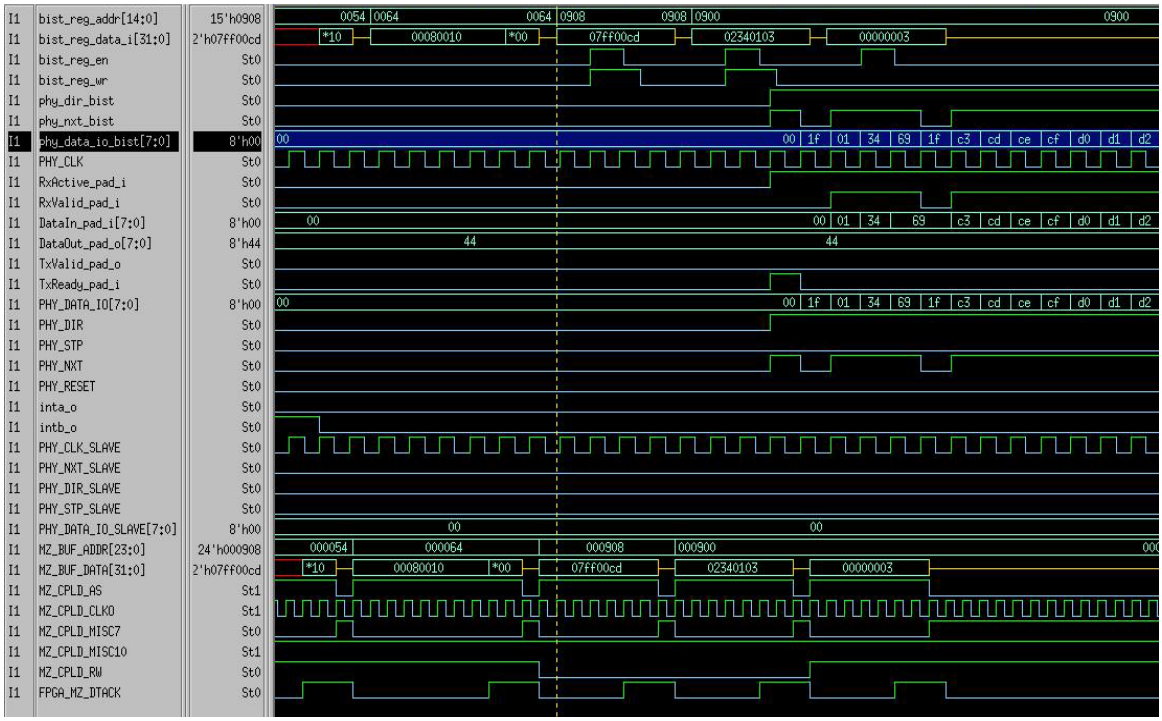


Figure 4.9 Status Stage Operation: Out Token and Data Token

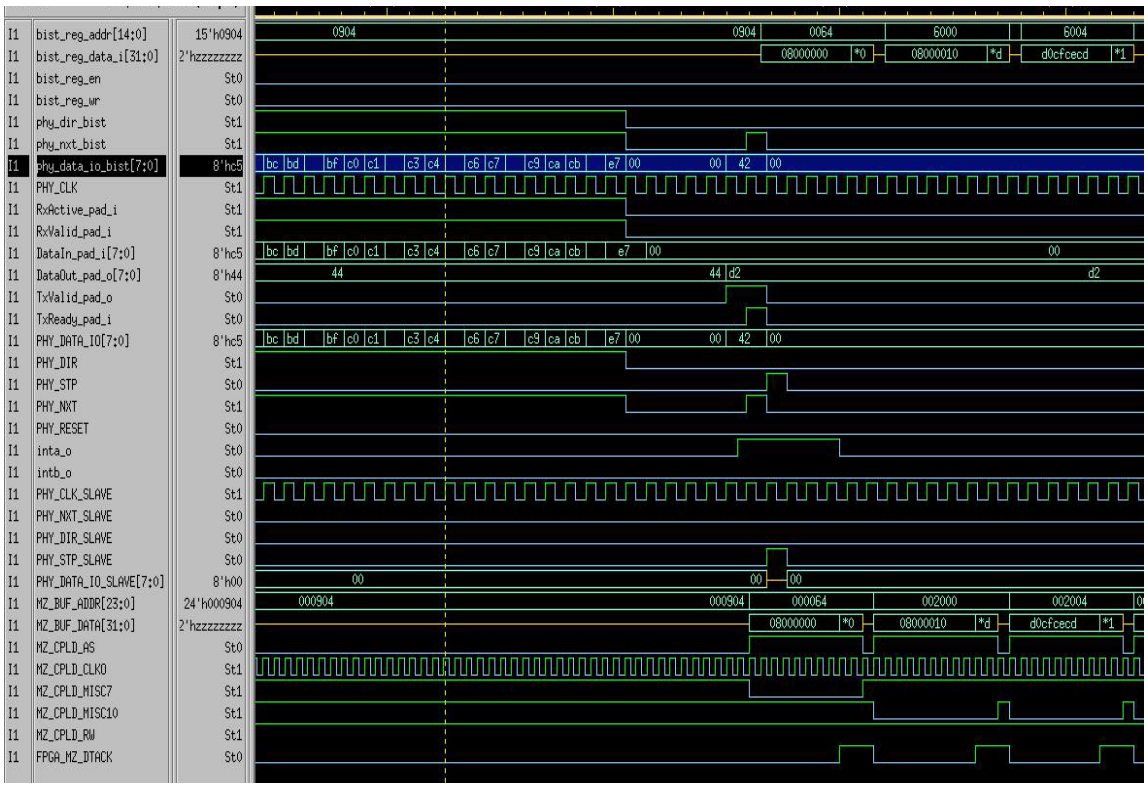


Figure 4.10 Status Stage Operation: Data Token and Ack Token

4.5 SIMULATION RESULT OF USB HOST TO USB SLAVE TRANSFER

Figure 4.11 and 4.12 shows how the USB host transfers data to USB slave. In the simulation, ULPI self test logic acts as the USB host.

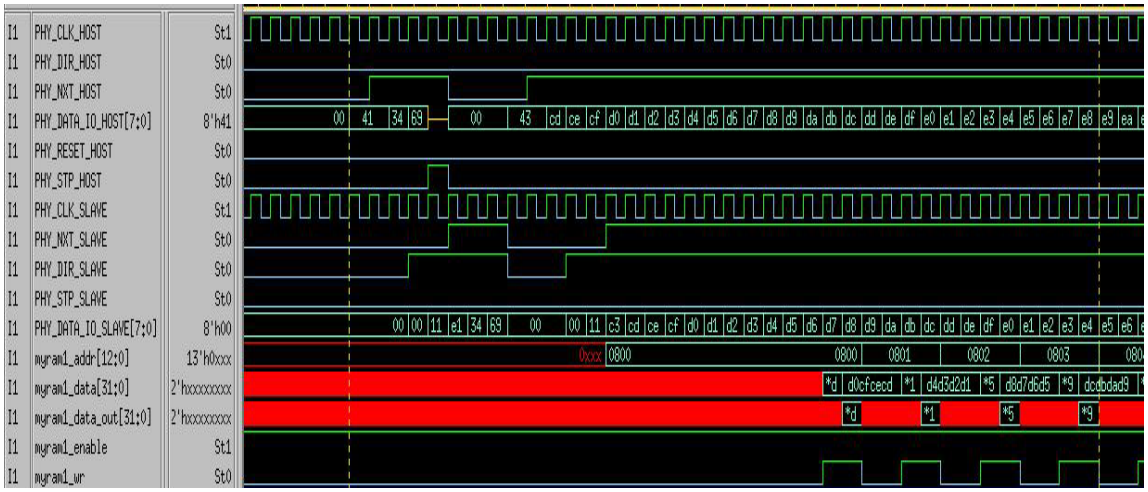


Figure 4.11 Host to Slave Transfer: OUT Token & DATA Token

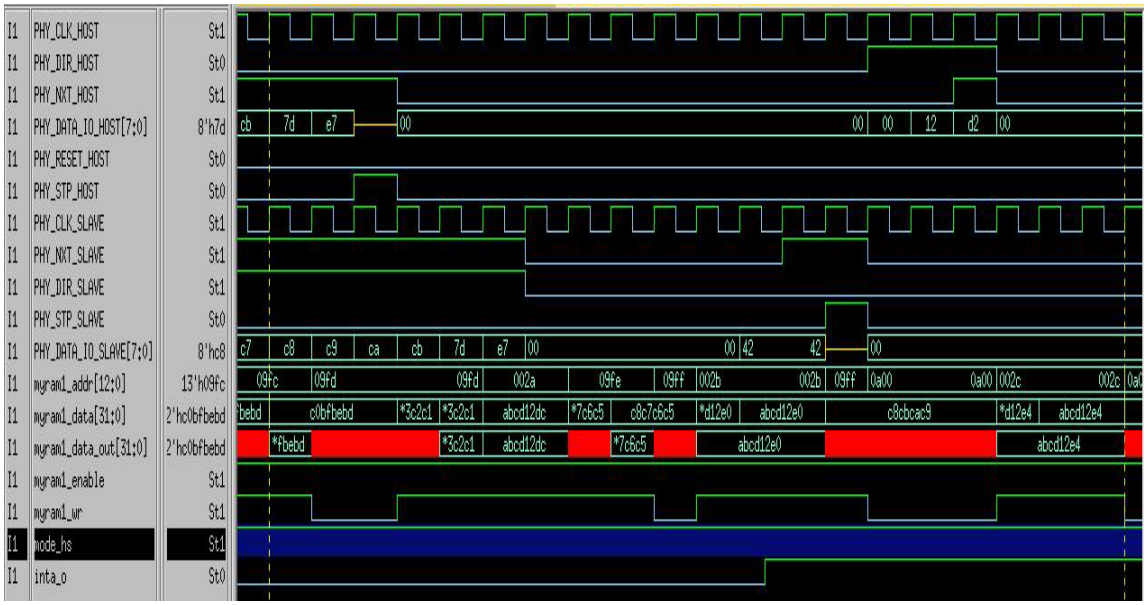


Figure 4.12 Host to Slave Transfer: DATA Token & Ack Token

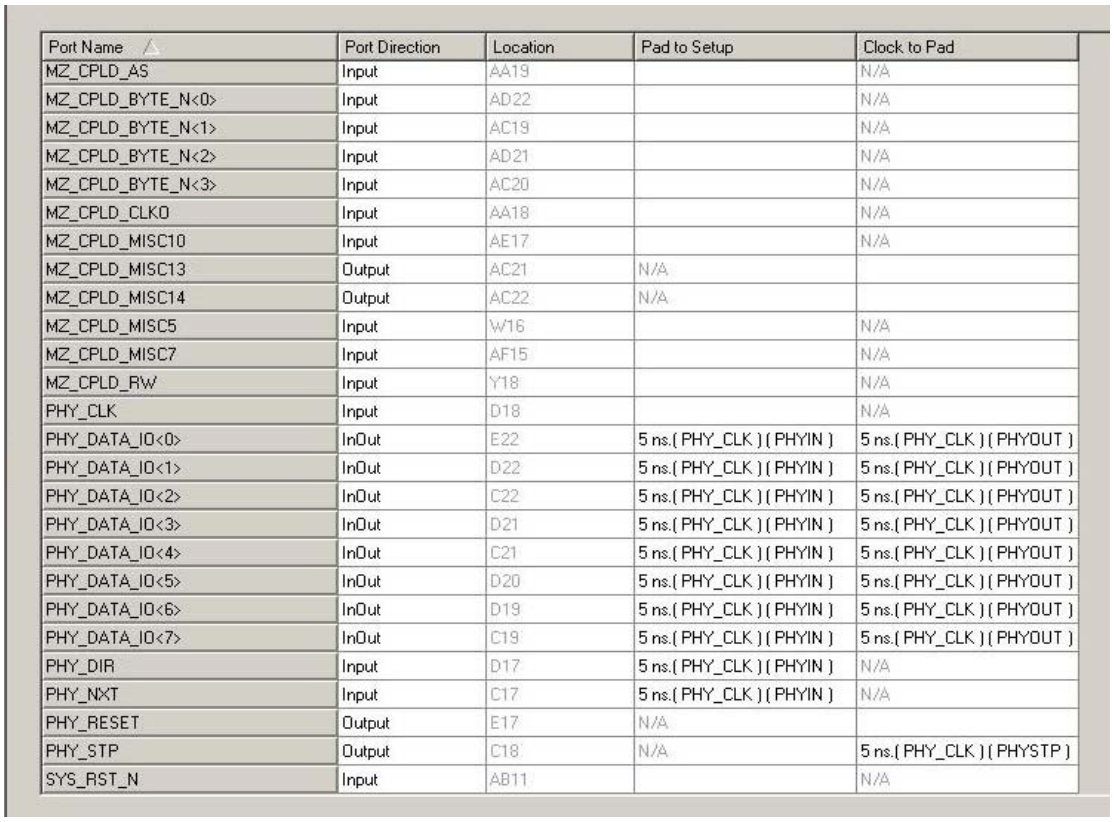
Chapter 5 Design Result

5.1 SYNTHESIS RESULT

Table 5.1 is the list of the design files being synthesized in the LL5000 Baseboard. Figure 5.1 shows how the PHY I/O ports of the design mapped to the FPGA pins.

Table 5.1 USB Design Verilog Files

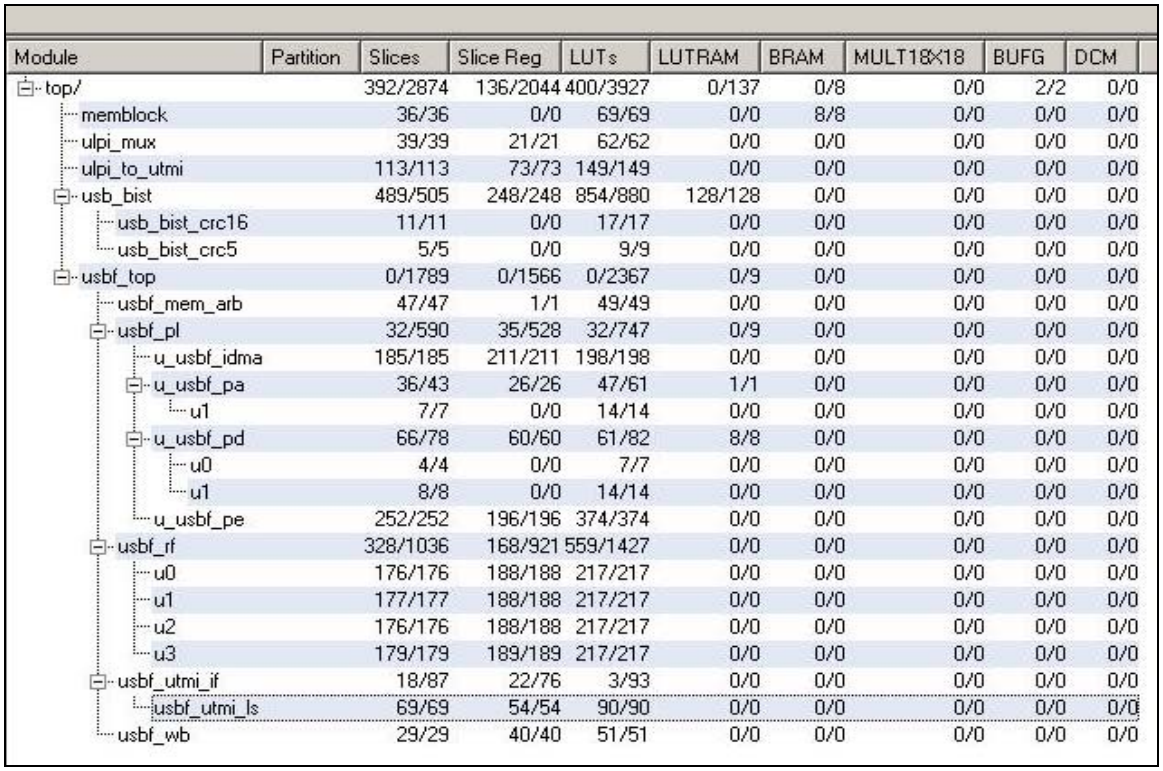
Design Hierarchy	Filename	Description
TOP	top.v	File also contains design for handing mezzanine bus access
UTMI-to-ULPI Translator	ulpi_to_utmi.v	See Section 3.2.1
ULPI Test Generation	ulpi_mux.v	See Section 3.2.2
ULPI Mux	ulpi_mux.v	See Section 3.2.3
UTMI Core	usbf_wb.v	Wishbone Logic
	usbf_mem_arb.v	Memory arbiter to resolve arbitration as PHY and Wishbone interface can both access the external memory.
	usbf_idma.v	Handle DMA operation. Function not used in this project.
	usbf_utmi.v, usbf_utmi_ls.v	UTMI state machine for design to enter different USB function such as FS mode, HS mode, SUSPEND etc.
	usbf_pa.v,usbf_pd.v, usbf_pe.v, usbf_pl.v	UTMI protocol layer engine for packing and unpacking UTMI protocol.
	usbf_ep_rf.v, usbf_ep_rf_dummy.v, usbf_rf.v	Register file design for endpoint registers
	usbf_crc5.v usbf_crc16.v	CRC 5 bits and 16 bits logic
External Memory	memory.v	Contains memory model to instantiated memory design from the Xilinx FGPA.
Timing Constraint	top.ucf	



Port Name	Port Direction	Location	Pad to Setup	Clock to Pad
MZ_CPLD_AS	Input	AA19		N/A
MZ_CPLD_BYTE_N<0>	Input	AD22		N/A
MZ_CPLD_BYTE_N<1>	Input	AC19		N/A
MZ_CPLD_BYTE_N<2>	Input	AD21		N/A
MZ_CPLD_BYTE_N<3>	Input	AC20		N/A
MZ_CPLD_CLK0	Input	AA18		N/A
MZ_CPLD_MISC10	Input	AE17		N/A
MZ_CPLD_MISC13	Output	AC21	N/A	
MZ_CPLD_MISC14	Output	AC22	N/A	
MZ_CPLD_MISC5	Input	W16		N/A
MZ_CPLD_MISC7	Input	AF15		N/A
MZ_CPLD_RW	Input	Y18		N/A
PHY_CLK	Input	D18		N/A
PHY_DATA_IO<0>	InOut	E22	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<1>	InOut	D22	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<2>	InOut	C22	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<3>	InOut	D21	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<4>	InOut	C21	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<5>	InOut	D20	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<6>	InOut	D19	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DATA_IO<7>	InOut	C19	5 ns.(PHY_CLK) (PHYIN)	5 ns.(PHY_CLK) (PHYOUT)
PHY_DIR	Input	D17	5 ns.(PHY_CLK) (PHYIN)	N/A
PHY_NXT	Input	C17	5 ns.(PHY_CLK) (PHYIN)	N/A
PHY_RESET	Output	E17	N/A	
PHY_STP	Output	C18	N/A	5 ns.(PHY_CLK) (PHYSTP)
SYS_RST_N	Input	AB11		N/A

Figure 5.1 IO Port Mapping on FPGA Pins

Gate counts of the design are summarized in Figure 5.2. The final design does not contain any latches.

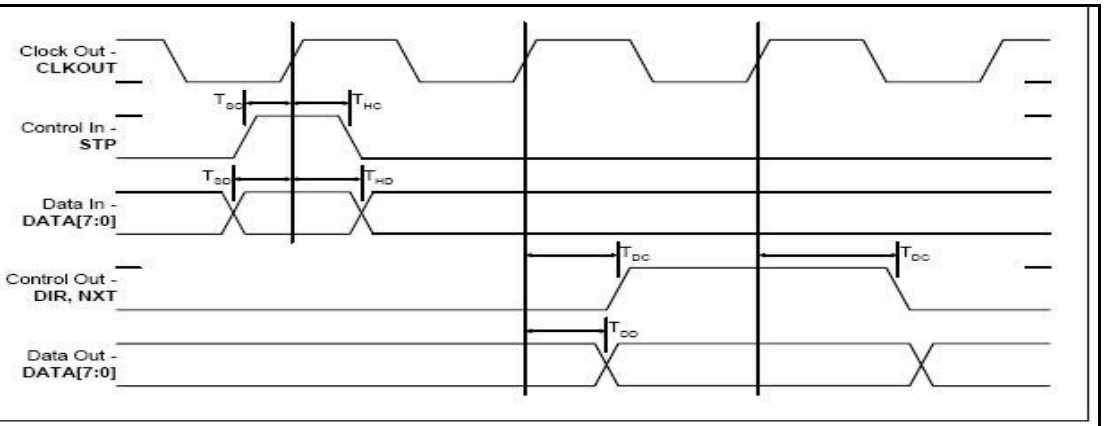


Module	Partition	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
top/		392/2874	136/2044	400/3927	0/137	0/8	0/0	2/2	0/0
memblock		36/36	0/0	69/69	0/0	8/8	0/0	0/0	0/0
ulpi_mux		39/39	21/21	62/62	0/0	0/0	0/0	0/0	0/0
ulpi_to_utmi		113/113	73/73	149/149	0/0	0/0	0/0	0/0	0/0
usb_bist		489/505	248/248	854/880	128/128	0/0	0/0	0/0	0/0
usb_bist_crc16		11/11	0/0	17/17	0/0	0/0	0/0	0/0	0/0
usb_bist_crc5		5/5	0/0	9/9	0/0	0/0	0/0	0/0	0/0
usbf_top		0/1789	0/1566	0/2367	0/9	0/0	0/0	0/0	0/0
usbf_mem_arb		47/47	1/1	49/49	0/0	0/0	0/0	0/0	0/0
usbf_pl		32/590	35/528	32/747	0/9	0/0	0/0	0/0	0/0
u_usbf_idma		185/185	211/211	198/198	0/0	0/0	0/0	0/0	0/0
u_usbf_pa		36/43	26/26	47/61	1/1	0/0	0/0	0/0	0/0
u1		7/7	0/0	14/14	0/0	0/0	0/0	0/0	0/0
u_usbf_pd		66/78	60/60	61/82	8/8	0/0	0/0	0/0	0/0
u0		4/4	0/0	7/7	0/0	0/0	0/0	0/0	0/0
u1		8/8	0/0	14/14	0/0	0/0	0/0	0/0	0/0
u_usbf_pe		252/252	196/196	374/374	0/0	0/0	0/0	0/0	0/0
usbf_rf		328/1036	168/921	559/1427	0/0	0/0	0/0	0/0	0/0
u0		176/176	188/188	217/217	0/0	0/0	0/0	0/0	0/0
u1		177/177	188/188	217/217	0/0	0/0	0/0	0/0	0/0
u2		176/176	188/188	217/217	0/0	0/0	0/0	0/0	0/0
u3		179/179	189/189	217/217	0/0	0/0	0/0	0/0	0/0
usbf_utmi_if		18/87	22/76	3/93	0/0	0/0	0/0	0/0	0/0
usbf_utmi_ls		69/69	54/54	90/90	0/0	0/0	0/0	0/0	0/0
usbf_wb		29/29	40/40	51/51	0/0	0/0	0/0	0/0	0/0

Figure 5.2 Design Utilization Summary

```
Design Summary
--------------------
Number of errors:      0
Number of warnings:    2
Logic Utilization:
  Number of Slice Flip Flops:          2,044 out of  26,624    7%
  Number of 4 input LUTs:              3,824 out of  26,624   14%
Logic Distribution:
  Number of occupied Slices:           2,425 out of  13,312   18%
    Number of Slices containing only related logic:   2,425 out of   2,425 100%
    Number of Slices containing unrelated logic:          0 out of   2,425   0%
     Total Number of 4 input LUTs:       3,927 out of  26,624   14%
    Number used as logic:             3,687
    Number used as a route-thru:        103
    Number used for Dual Port RAMs:     128
      (Two LUTs used per Dual Port RAM)
    Number used as Shift registers:       9
  Number of bonded IOBs:                 92 out of     487   18%
  Number of RAMB16s:                     8 out of      32   25%
  Number of BUFGMUXs:                    2 out of       8   25%
```

The timing requirement of PHY DATA and Control signals are listed in Figure 5.3. After the place & route stage, the final design is not able to meet the timing requirement on PHY_DATA and PHY_STP. Even though the design has no other logic after the gated output of these two signals, the design stills fail to meet the timing constraint. The timing path delay for PHY_DATA and PHY_STP are shown in Figure 5.4; both of the signals miss the setup time constraint by about 1.7ns.



PARAMETER	SYMBOL	MIN	MAX	UNITS
Setup time (control in, 8-bit data in)	T_{SC},T_{SD}	5.0		ns
Hold time (control in, 8-bit data in)	T_{HC}, T_{HD}	0		ns
Output delay (control out, 8-bit data out)	T_{DC}, T_{DD}	2.0	5.0	ns

Figure 5.3 Timing Requirement from SMSC USB3300 Specification[5]

Figure 5.4 Time Delay on PHY DATA and PHY STP

5.2 ANALYSIS OF DESIGN RESULT

The final design fails to transmit the whole packet from the USB host to the USB slave. However, the design is able to execute and complete the build in self test. This validates the design can generate read and write command to ULPI PHY registers. It also validates the UTMI Core logic can generate Endpoint buffer full and Endpoint buffer empty interrupt accordingly. Figure 5.5 shows the execution of self-test In-Token and Out-Token and some ULPI PHY registers read/write on the evaluation board.

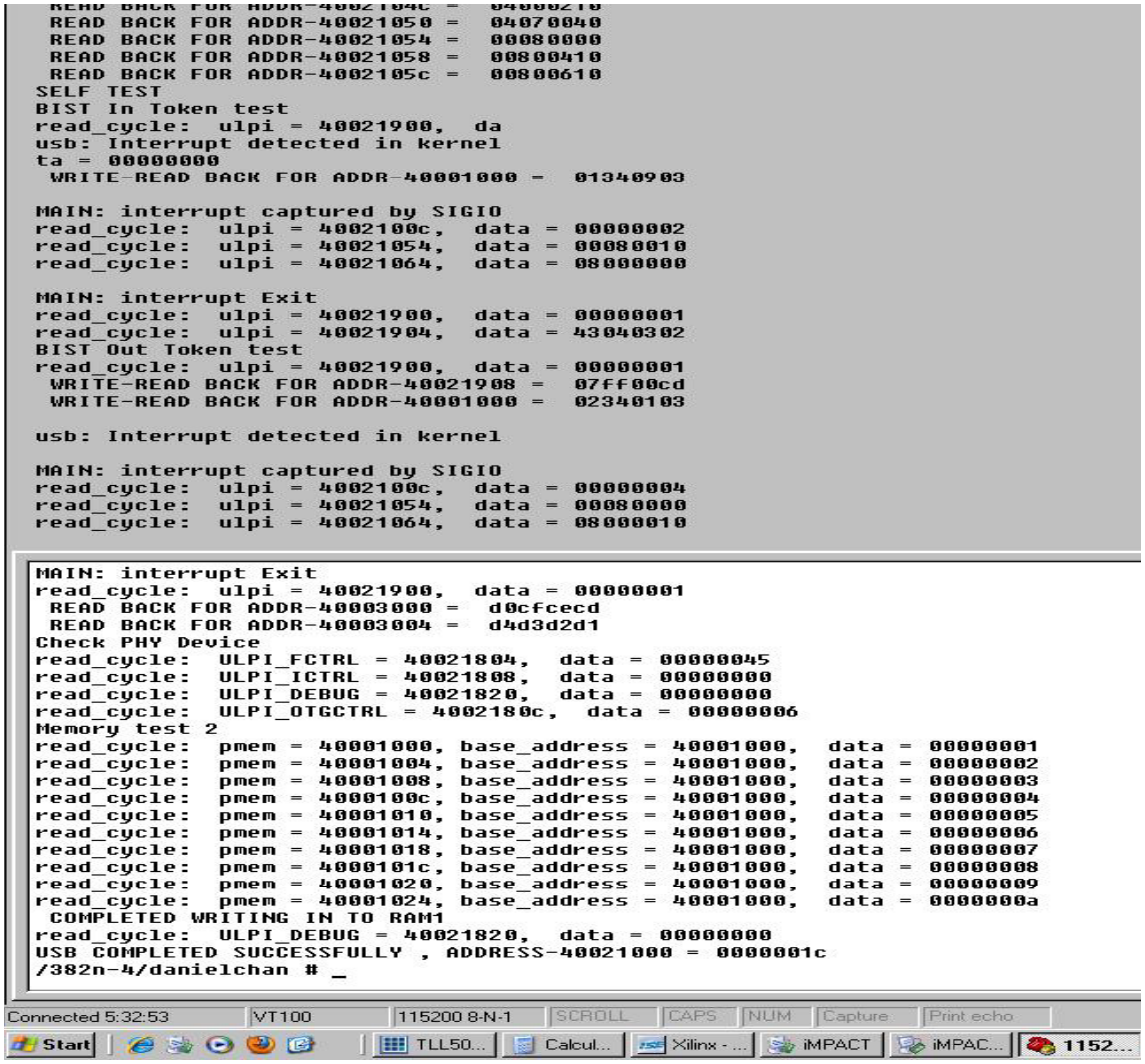


Figure 5.5 Validation Result from IN-Token and OUT-Token Self Test

To validate the design, a data stage sequence is executed in the system. This sequence is repeated multiple times. The design can detect the OUT Token consistently. However, package corruption is seen in both OUT Token and DATA Token sequence. Sometimes, the software execution also hangs. Since data are not reliably received, no bandwidth result is measured. Table 5.2 illustrates the interpretation of the data package.

Refer to Figure 2.2 and 2.3 on the Data Stage sequence and content. Figure 5.6 shows the expected result from simulation and actual result from one of the validation trial runs on the Baseboard.

Table 5.2 USB Data Stage Package for Validation

Package Sequence	Interpretation of the Package
OUT Token 0xe1, 0x34, 0x69	Token PID =0xe1 ADDR, ENDP, CRC=0x34,0x69
DATA0 Token 0xc3, 0xcd, 0xce, 0xcf, 0xd0…	Token PID =0xc3 DATA = 0xcd, 0xce, 0xcf, 0xd0
ACK Token	Not Shown. Package is not received properly.

Based on the validation result, the first package PID is consistently latched twice in both OUT Token sequence and DATA0 Token sequence. The timing problem on the PHY_DATA probably causes this problem. The package flip problem could be due to the cable noise and a combination of timing problems. The package drop seems to have a repeated sequence. This could be due to an out-of-sync problem in the transmission.

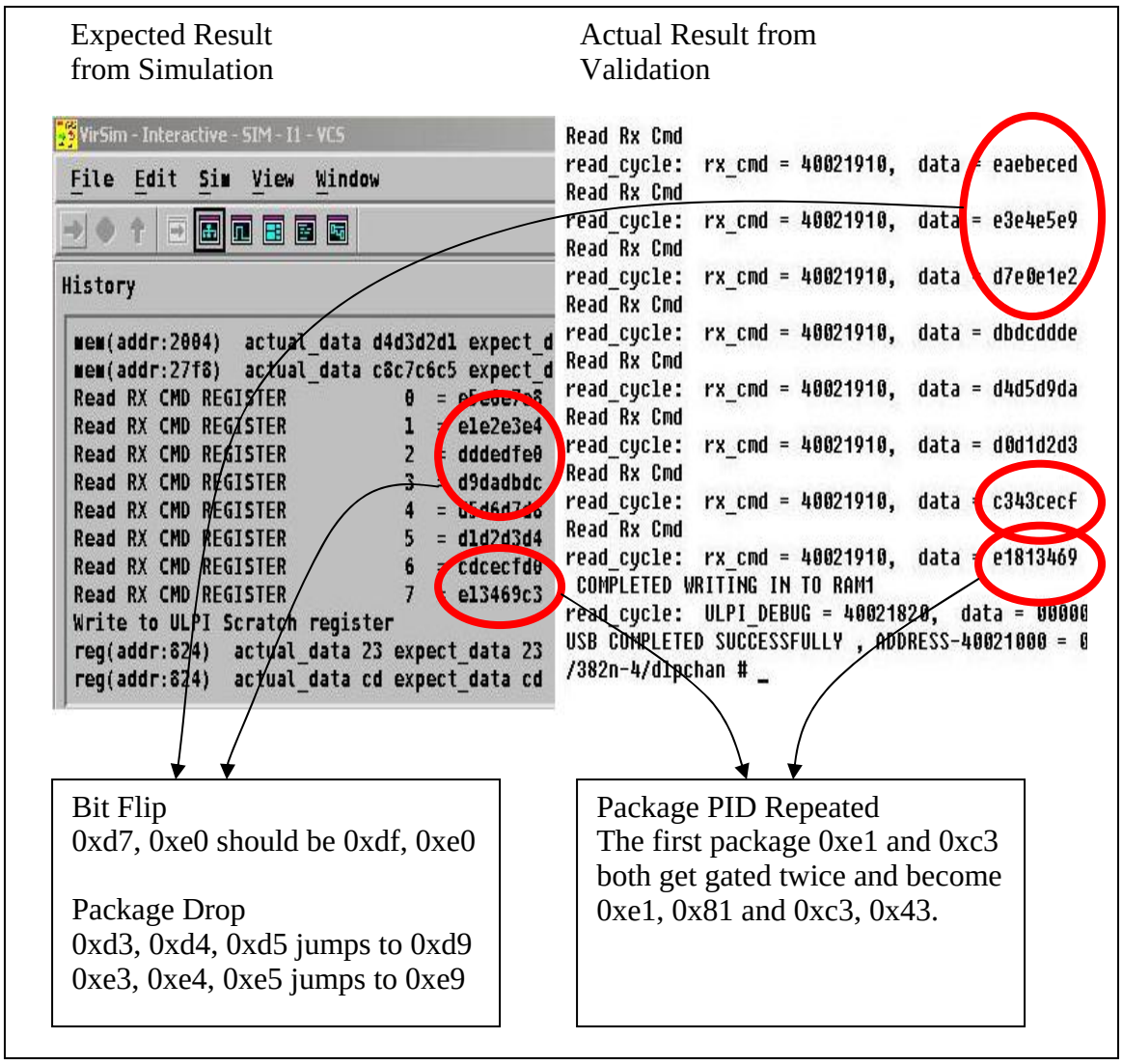


Figure 5.6 Expected and Actual Result from Baseboard

5.3 SOLUTION TO DESIGN PROBLEMS

The design needs to be modified in order to resolve the timing problem on PHY_DATA. One possible way is to remove the pre-fetch and gating logic on PHY_DATA and PHY_STP. We insert additional buffer on PHY_DATA and PHY_STP to miss the whole PHY_CLK cycle. Thus, the design can meet the setup timing

requirement for the ULPI PHY connector on PHY_DATA and PHY_STP. Table 5.3 suggests solutions for the problems seen in the Baseboard's validation.

Table 5.3 Solution to Design Problems

Problem	Solution
First Package Flip	Fixing the timing problem should resolve this issue.
Package Flip	If the problem still occurs consistently, it could be due to cable noise issue. A software solution can be applied to correct the flipped bit. For examples, we can add an error correcting package before the CRC16 or implement a best-2-out-of-3 system.
Package Drop	We can shorten the number of data bytes transmitted in every data token.

References

[1] "USB in a Nutshell," www.beyondlogic.org/usbnutshell

[2] R Usselmann. "USB 2.0 Function IP Core," 2002. www.opencores.org

[3] "UTMI+ Low Pin Specification (ULPI) Revision 1.1" Oct 20, 2004. www.ulpi.org/documents.html

[4] TLL5000 User Guide, v1.3." The Learning Labs, Inc., Web. 1 Nov 2009. <http://users.ece.utexas.edu/~mcdermot/arch/labs/DOCUMENTATION/TLL5000_User_Guide_Ver1.3.pdf>

[5] "SMSC USB 3300 Specification Revision 1.0.8" Nov 07, 2007

Vita

Iat Pui Chan completed high school in Macau SAR, China. After high school, he pursued his college education in the US. In 1999, he earned his bachelor degree in electrical engineering from the University of Texas at Austin. Currently, he lives in Austin and works for Marvell Inc. as a design verification engineer.

Permanent address: 9012 Corran Ferry Dr, Austin TX 78749

This report was typed by the author.